Exploiting Large Datasets Improves Accuracy Estimation for Multiple Sequence Alignment

Luis Cedillo, Hector Richart Ruiz, Dan DeBlasio
Computer Science Department, University of Texas at El Paso

Abstract

Multiple sequence alignment plays an important role in many important analyses. However, aligning multiple biological sequences is a complex task, thus many tools have been developed to align sequences under a biologically-inspired objective function. But these tools require a user-defined parameter vector, which if chosen incorrectly, can greatly impact downstream analysis. Parameter Advising addresses this challenge of selecting input-specific parameter vectors by comparing alignments produced by a carefully constructed set of parameter configurations. Ideally accuracy would be used to rank alignments, but in practice, we do not have a reference from which accuracy is calculated. Thus, it is necessary to *estimate* the accuracy in order to rank alignments. The accuracy estimator `Facet` computes an estimate of accuracy as a linear combination of efficiently-computable feature functions. In this work we introduce two versions of `Lead` (short for Learned accuracy estimator from large datasets) which use the same underlying feature functions as `Facet` but are built on top of highly efficient machine learning protocols, allowing us to take advantage of a larger training corpus. This produces an estimator that is more correlated with accuracy. For Parameter Advising, `Lead` shows an increase of 6% on testing data over using only the default parameter vector.

1 Introduction

For many computational biologists, aligning multiple biological sequences, mainly protein sequences, is an essential step in analysis. These alignments provide key insights into the correlation of various regions in the protein, and can be a guide to determining evolutionary history. From a computational standpoint though the problem is complicated, and in fact, it was proven that finding an optimal alignment of a set of sequences is NP-Complete [32, 16]. Because of this juxtaposition of the importance and complexity of the problem there are many existing heuristics that find *good* (though not optimal) multiple sequence alignments [28, 27, 13, 10, 23, 33, among others]

None of these tools address a fundamental issue: the multiple sequence alignment problem requires as part of its input a user-defined objective function. While there are several well-studied objectives, the most common of which is sum-of-pairs, this still leaves the open question of the function's parameters: the rewards and penalties associated with each operation used to construct the alignment. Thus when new users, or even experienced users with a new class of inputs, approach the task of creating a multiple sequence alignment, they are required to choose a value for each of the tunable parameters of the tool (we call the collection of parameter values used is called a *parameter vector*). Making the *wrong* choice can have a highly detrimental impact on downstream analysis; manually tuning these algorithm values in a systematic manner requires not only a more-than-cursory understanding of the underlying application and the data, but also substantial amounts of time. Because of this most users rely upon the *default* parameter vector included with each tool. These defaults are specifically set to provide good results on average across all types of input, but the most interesting biological datasets are typically far from average.

Most alignment objective functions, and in turn the tools that utilize them, have parameters for both the penalty assessed for adding gaps into a sequence and the penalty/reward for the substitution (match/mismatch) of any two amino acids (residue). This means for the 20-character amino acid alphabet used for protein sequences, 190 individual values must be specified by the user for substitutions alone. Thankfully this is a well-studied problem, and there

are many highly-accurate pre-calculated databases of substitution scores, called *substitution matrices*. Typically users are given a choice from a class of matrices like `VTML` [22], `BLOSUM` [12], or `PAM` [6]. Within each class, there are different matrices available that are optimized for sequences with a specific level of entropy. When substitution matrices are used, the total number of tunable parameters is greatly reduced, simplifying (though not eliminating) the parameter choice problem.

One confounding factor for making an informed parameter vector choice for multiple sequence alignment is that unlike some domains, such as transcript assembly which has a readily available quality measurement which can be used on any input [9], the standard method for assessing accuracy requires comparison to a reference (ground-truth) alignment. Thus, the accuracy of a computed alignment can only be measured on so-called *benchmark* sequence sets. Typically these benchmarks are constructed by aligning the three-dimensional structure of proteins to find amino acids that are highly correlated. Those groups of amino acids (one from each protein) that are within some threshold of distance in the 3D alignment are turned into complete columns of the sequence alignment. These anchoring positions are called *core columns*. Accuracy of a computed alignment is then calculated as the fraction of substitutions from the core columns of the reference alignment that are recovered. Several databases of these benchmarks exist and are regularly used to compare multiple sequence alignment tools [3, 11, 26, 30, 2].

When a new (non-benchmark) sequence set is presented and aligned, there is no reference alignment from which to measure accuracy. In this case, one is left to estimate the accuracy of the computed alignment. There are a variety of tools available to compute this estimation, which can generally be divided into two main categories: *scoring-function-based* tools [14, 4, 29, 1] which calculate a value based on the combination of measurable attributes of the alignment alone; and *support-based* tools [20, 19, 25] which use a collection of alignments over the same sequences to label each one in the set.

Using one of the tools above to estimate accuracy, a user can then begin to search for the optimal parameter vector for their input. While this can be done by hand or using an off-the-shelf iterative optimization mechanism [such as coordinate ascent; 34], this is typically still quite time consuming. To automate this process DeBlasio and Kececioglu [8] developed the *Parameter Advising* framework, which without additional wall-clock time if the appropriate resources are available can make input-specific parameter choices for multiple sequence alignment (details of an advisor are contained in Section 2 to make this work fully self-contained). In that work, they show the advisor using the `Opal` alignment tool and the `Facet` accuracy estimator proved to have the highest advising accuracy.

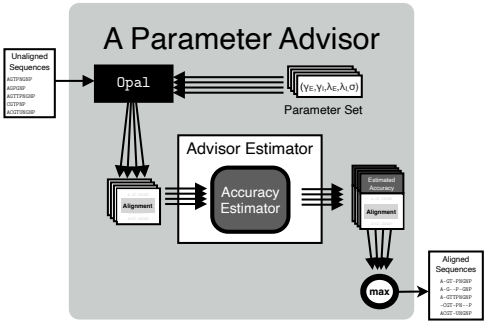

Figure 1: The parameter advising process.

In this study, we present a new estimator `Lead` (short for Learned accuracy estimator from large datasets) in two variants: `Lead-MLP` (which uses a multi-layer perceptron) and `Lead-LR` (which uses linear regression). These two new scoring-function-based accuracy estimators which reimagine the `Facet` estimator by using modern machine learning techniques for optimization, rather than combinatorial optimization, to exploit the much larger datasets we have produced. Although `Lead-MLP` and `Lead-LR` rely on the same underlying set of efficiently-computable feature functions as `Facet`, their parameter advising performance is substantially better. They show an average increase in advising accuracy of 6% over using only the default parameter choice, this is more than a 2% increase over `Facet` on estimator-aware advisor sets. The newly developed estimators also show a much stronger correlation with true accuracy due to the fact that we have been able to optimize for this task specifically.

2 Summary of Prior Work

Parameter Advising, depicted in Figure 1, takes the same input as the underlying scientific application and returns a single result. So to the end-user it appears to be just another tool to solve an existing problem but in this case a tool without any tunable

parameters. But the process of selecting a parameter vector that provides higher-than-default accuracy, tailored to user input sequences, is abstracted away.

An advisor contains two key components: a set of candidate parameter vectors, called an *advisor set*; and an accuracy estimation tool used to choose from among those vectors, called an *advisor estimator*. The actual construction of these two components is described in detail below.

The Parameter Advising procedure is as follows:

- The input is combined with each of the parameter vectors in the advisor set (and the underlying application) to create a set of *candidate output* (alignments).
- The advisor estimator is then used to label each of the candidate alignments with an estimated accuracy.
- Finally, using the labeled estimated accuracy the candidate alignment that is predicted to be most accurate is returned to the user, therefore selecting the parameter vector.

Parameter advising is an *a posteriori* selection method and may not work in all cases, but in those applications where resources can be allocated such that each of the candidate outputs and their labels can be produced in parallel, the advising procedure adds only a negligible amount of wall-clock time (the time needed to run the estimator, also in parallel, and select the best output). In addition to multiple sequence alignment, this framework has been shown to work well for selecting parameters for transcript assembly [9].

Throughout the paper it is assumed that there is a consistent set B of benchmark sequence sets (the precise sets themselves will be explained in Section 3.3). Because there is bias in the sampling of these benchmarks (as they are made by hand, additional commentary on this in later sections), it is assumed that at all stages each benchmark will have a weight (or importance), w_b for $b \in B$. This weighting will be used to correct for the overabundance of some types of examples.

2.1 Advisor Sets

For the task of parameter advising an advisor set need to satisfy two main criteria: the set should be small to reduce resource consumption, and should contain at least one parameter vector that will work well on each given input. Due to the somewhat contradictory nature of these two ideas, a method is needed to determine the balance between competing interests. One way to do this is to put an actual limit on the resource consumption by specifying k, the number of parameter vectors to include in our set. Experiments can be used to determine the cardinality at which there are diminishing returns, meaning adding more computational power does not provide substantial gains in advising accuracy.

Existing methods have been developed to accomplish the task of finding such sets with restricted cardinality. They all rely on first enumerating the entire universe U of parameter vectors for a tool across the entire set of benchmarks B. The combination of a benchmark $b \in B$ and a parameter from the universe $p \in U$ produces an alignment, $\mathbb{A}_{bp}$. Because the reference alignment is known the true accuracy, A_{bp}, can be calculated. If there is a pre-existing accuracy estimator it is assumed that the estimated accuracy for each of these alignments is E_{bp}.

The *oracle* set finding method uses only the true accuracy, and finds sets that are optimal if you had an estimator function that predicted accuracy exactly (an oracle). Finding optimal sets while taking the accuracy estimator into account is not practical, thus a *greedy approximation* method was developed that works well in practice.

2.1.1 Oracle Sets

While finding optimal parameter sets is NP-Complete [7] whether you have the estimator values or not. For a small enough universe the Oracle set problem can be solved exactly using an integer linear program (ILP).[1] The program contains variables s_p which indicates if parameter p is in the advisor set, and variables v_{bp} that will signify if the advisor using the found set chooses parameter p for benchmark b. Three groups of constraints then ensure that:
(1) for each benchmark has only one chosen align-

[1] As is noted in Section 4 for the much larger universe used in most of the study is it not possible to solve these problems exactly at the time of publication so results are shown on the smaller parameter universe from previous studies for only the Oracle sets.

ment,

$$\sum_{p \in U} v_{bp} = 1$$

(2) an alignment is only chosen if the associated parameter is chosen,

$$v_{bp} \leq s_p$$

and (3) that only k parameters are chosen.

$$\sum_{p \in U} s_p \leq k$$

With the constraints in place, the objective then would be to maximize the total accuracy of all of the chosen alignments

$$\sum_{b \in B} \sum_{p \in U} v_{bp}(w_b A_{bp}).$$

The ILP then contains $O(|U||B|)$ variables and constraints, though only $O(|U|)$ of the variables need to be explicitly declared as binary (integer), if the parameter variables are binary the alignment variables must be binary valued for a solution to be optimal (with the exception of ties in accuracy, but this would not impact optimality).

2.1.2 Greedy Sets

Finding optimal estimator-aware sets for large cardinalities is impractical, but a greedy procedure has been shown to provide substantial accuracy increases (over Oracle sets) in practice. Assume that an advisor $\mathcal{A}_\delta(P)$ takes as input the set of parameter vectors $P \subseteq U$, and returns the average advising accuracy (over all benchmarks in B) of a parameter advisor that uses P as its advisor set (it has access to the set of benchmarks, true accuracies, and estimated accuracies). For generalization purposes during training (but not for showing the results in later sections), a margin of error on the estimator, δ, is used within the function $\mathcal{A}$. Rather than using a single alignment's accuracy for a benchmark (the one with highest estimated accuracy), the method calculates the average for all alignments of a benchmark that are within δ of the maximum [details of this can be found in 7]. The initial set P_1 is the Oracle set of size 1, in other words the best single default parameter vector. At each step the greedy procedure finds the *next best* parameter to add that is not already in the set, i.e. the one that when added provides the highest advisor accuracy. That is to say the greedy method at iteration i first finds

$$p_i = \underset{p' \in U \setminus P_{i-1}}{\arg\max} \left\{ \mathcal{A}_\delta \left(P_{i-1} \cup \{p'\} \right) \right\},$$

then sets $P_i = P_{i-1} \cup \{p_i\}$. It continues with this procedure until P_k is found.

2.2 Advisor Estimator

It is the advisor estimator's responsibility to rank the candidate outputs in such a way that the highest-ranked alignment is the most accurate. The best estimator is referred to as the Oracle (also used above) which knows the true accuracy and would be able to make a ranking based on this information. In the absence of an Oracle something close needs to be found. There are in theory two types of advisor "estimators": one that assigns an estimated accuracy to each candidate (as will be used here), and another that simply chooses from the set of candidates. This second type of advisor is yet to be explored, but would still likely need to rely on some way to extract information (features) from the alignment itself in order to make a judgment.

2.2.1 Feature Functions

The feature functions used were heavily studied in previous work, they consist of efficiently-computable functions that extract a singular numeric value that is correlated with accuracy. This set contains both canonical feature functions that have been historically used for benchmarking as well as some developed specifically for this task. Because the focus is on *protein* multiple sequence alignments, many of the feature functions will exploit the predicted secondary structure of the sequences in the alignment to calculate a value. This means each amino acid in the alignment is labeled with both a class (α-helix, β-strand, or coil) and a set of three probabilities which the features utilize.

The features developed are also non-local, meaning they generally use information from across the entire alignment. This means that they contain more information than an alignment objective function (which by design are local calculations).

Details of the feature computations can be found in Kececioglu and DeBlasio [14], but they are listed here for convenience as they are a crucial component of our new methods, roughly in order of importance:

- **Secondary Structure Blockiness** — maximum percentage of an alignment that can be covered by a packing of contiguous two-dimensional blocks of residues with the same secondary structure class label.

- **Percent Identity** and **Secondary Structure Percent Identity** — the fraction of aligned amino acids (structure class labels) in all extracted pairwise alignments from the input that are precisely the same.

- **Average Replacement Score** — average scaled BLOSUM62 score for all aligned amino acids from the induced pairwise alignments.

- **Gap Extension Percentage** and **Gap Open Percentage** — percentage of characters in the alignment that are (runs of) gap characters.

- **Information Content** — average over all columns of the difference in distribution of amino acid frequencies compared to a background distribution.

- **Substitution Compatibility** and **Gap Compatibility** — fraction of pairs of columns that pass the 4 gametes test when converted into 1 and 0 by use of majority/minority amino acid character or gap/non-gap character.

- **Structure Agreement** — weighted sum of structure class probabilities for a window around each aligned pair of amino acids in induced pairwise alignments.

- **Gap/Coil Density** — fraction of gap characters that align with the coil structure class label.

- **Core Column Density** — fraction of columns that are gap-free and contain mostly the same amino acid.

- **Sequence Consensus** and **Gap Consensus** — fraction of pairs of columns that have the same majority/minority amino acid character or gap/non-gap character pattern.

All feature values are normalized to be in the range $[0, 1]$, and if the value cannot be defined (i.e. gap open percentage when there are no gaps) the value is fixed at 0.

2.2.2 The `Facet` Estimator

The `Facet` (short for Feature-based Accuracy Estimator) model uses a linear combination of the feature values to calculate an accuracy estimate. Given a set of weights $T = (t_1, t_2, ..., t_n)$ the `Facet` value is computed from a set of feature functions $F_1, F_2, ..., F_n$ as

$$L_T(\mathbb{A}) := \sum_{1 \leq i \leq n} t_i F_i(\mathbb{A}).$$

There are several methods that can be used to learn an accuracy estimator, one can either try to match the estimator values to the true accuracy exactly or, what worked better in previous experiments for the task of parameter advising for multiple sequence alignment, focus on the objective of ranking candidates.

All of the `Facet` results shown will use the methods described in DeBlasio and Kececioglu [8], which learns the values of T using combinatorial optimization (in this case linear programming, LP). In this case for all pairs of alignments $\mathbb{A}_{bp}$ and $\mathbb{A}_{bq}$ aligning the same benchmark using parameters p and q such that $A_{bp} > A_{bq}$, the goal is to minimize the ranking error:

$$\max\left\{\left(\left(A_{bp} - A_{bq}\right) - \left(L_T\left(\mathbb{A}_{bp}\right) - L_T\left(\mathbb{A}_{bq}\right)\right)\right), 0\right\}.$$

Note that in this case, the only free variables are T, because the alignments are fixed then the feature values and true accuracies are also. Additional constraints are added to ensure that the estimator as well as the values in T are in the range $[0, 1]$.

3 Methods

3.1 `Lead-MLP`: Exploiting Non-Linearity

3.1.1 Learning hyperparameters

To solve the regression problem discussed earlier, we trained a multi-layer perceptron (MLP). Network architectures are sometimes inspired by the underlying structure of the input data [21], but in situations where this is not feasible, we rely on guided search to find the most appropriate architecture. It can be time-consuming and data-intensive to train a network with too many parameters. It can also make exploring different architectures computationally expensive. The key is to strike the right balance between accuracy, complexity, computation resources,

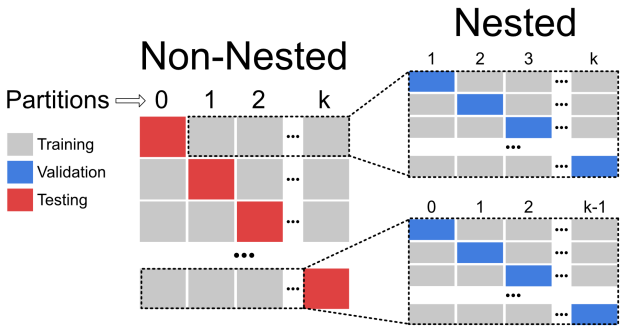

Figure 2: Visual depictions of both cross-validation mechanisms.

and generalization. To this end, we excluded certain architectures from our search space, as is common in most works. For this regression task, we found that large and deep MLP architectures resulted in poor performance in terms of both resource use and accuracy scores. Nonetheless, our proposed method, which combines various techniques and experimentation, proved effective in identifying the most appropriate architecture for our specific task.

We conducted our experiments with the `scikit-learn` multilayer perceptron (`MLPRegressor`) model. The experiments were conducted in two stages that used different cross-validation methods to try and avoid overfitting the found model architecture. In the first stage we use nested cross-validation, and the second we use non-nested cross-validation (Figure 2). In all cases the training samples are initially split into k partitions, and in each of the k top-level instances one of the partitions is held out for testing, and the other are in the training set. In the nested case, a lower level split is made of the remaining $k-1$ partitions with one of these being used for *validation* and the other $k-2$ being used for testing. The first stage, the architecture search, was conducted using nested cross-validation, while to make the final model choice decision using as much data as possible, a second stage which also optimizes the batch size, was conducted using non-nested cross-validation.

The first stage of training used nested cross-validation for the local search of architecture, training each model three times with three different random seeds. We first randomly generated 64 unique architectures that balanced performance with the amount of computational resources required for training. These architectures were small in size, consisting only of two hidden layers and a maximum of 16 neurons per layer. All models at this stage were trained with a batch size value of 256 and ReLU as the activation function. Our local search algorithm explored these architectures iteratively and selected the one that minimized the average mean squared error on all $k-1$ validation sets. In this case, however, we used a modified mean squared error to account for the highly imbalanced distribution of the target variable. A threshold of 0.5 was used to binarize the continuous target data into binary values. We then applied the mean squared error function to these updated binary targets and their respective predictions. This variation of mean squared error was not optimized when training the MLP model itself. This variation was used to measure the final performance of the model on validation data. To further refine our search, we implemented early stopping criteria during training. This allowed us to stop training architectures that were not showing promising results on the validation set.

Having located the most suitable architecture through nested cross-validation, we refined the model by conducting another local search using the non-nested cross-validation process described earlier. During this refinement stage our goal was to find an optimized batch size using exhaustive search. The batch sizes tested were 32, 64, 128, 256, and 512. To prevent overfitting, we used several early stopping conditions and fixed random subsample as a validation set. As a final step, we selected the batch size that returned the model with the highest R-squared score on the training data. With this multifaceted approach, we were able to find the most suitable architecture for the MLP network, ensuring robustness and generalizability while taking into account performance and computational resources.

Two different values were used for the α hyperparameter, which is responsible for the L_2 regularization term's strength. The architecture search used an α value of 0.01, while the refinement stage used an α value of 0.1. When regularization is applied, a penalty term is added to the cost function to prevent overfitting. The lower α value used for the architecture search helped us easily discard overfitting-prone architectures, while a higher α value for the refinement stage allowed us to utilize the entire training corpus without overfitting when training the final model for that fold.

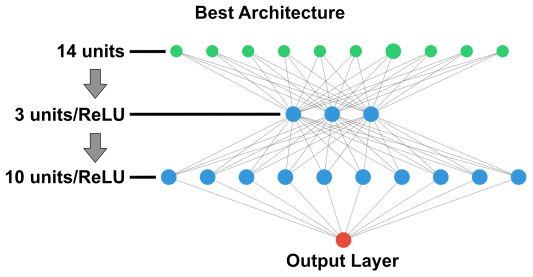

Figure 3: Visual depiction of the model architecture that was optimal for a majority of cross-validation folds.

3.1.2 Training the model

Figure 3 shows the topology that performed the best on 10 out of the 12 folds. After identifying an architecture and its batch size for each cross-validation fold, it was time to train the final model and compare its final performance with unseen data. A separate estimator was learned on the training set of each cross-validation fold. All results are reported as an average over all of the folds. A `MLPRegressor` was instantiated with `Adam` [18] as the optimizer, ReLU as the activation function, and the mean squared error as the loss function. Training was conducted with an adaptive learning rate of 0.1, which was lowered by a factor of 0.001 based on the behavior of the loss function. This adaptive learning rate allowed the training process to converge efficiently while avoiding overshooting the optimal solution. To help the model converge and make the decision surface smooth, it is commonly recommended to use a normalization method like `MinMax` or Standard Scaler when training MLPs, but due to the nature of our features, this preprocessing step was not necessary.

A note about implementation

All of the methods above as well as the experiments below were performed using `scikits-learn` [24]. All models, architectures, and training scripts (written for `python3`) for both `Lead-MLP` and `Lead-LR` are released on the DeBlasio Lab GitHub, at `github.com/deblasiolab/Lead`.

3.2 `Lead-LR`: Utilizing More Data

As mentioned in Section 2.2.2, `Facet` was trained to minimize the ranking error of the accuracy prediction made by the tool. This was done mainly because it showed better performance than trying to match the accuracy value directly. One reason this was likely the case is that the LP method used was not able to scale to large numbers of samples practically, and thus narrowing the task to the actual use was better with limited data. But, modern machine learning optimization tools now allow us to use much more data to train models, they also enable us to fully utilize complex hardware to accomplish this task.

Table 1: Example `Lead-LR` Feature Weights

Feature Function	Weight
Secondary Structure Percent Identity	1.232
Gap Open Percentage	−0.551
Secondary Structure Blockiness	0.475
Average Replacement Score	−0.455
Core Column Density	−0.311
Substitution Compatibility	−0.296
Gap Compatibility	−0.260
Gap Consensus	−0.147
Structure Agreement	−0.113
Gap/Coil Density	−0.072
Information Content	0.052
Percent Identity	−0.049
Gap Extension Percentage	0.015
Sequence Consensus	0.005

`Lead-LR` was implemented using the Linear Regression model of `scikits-learn`. A separate estimator was learned on the training set of each cross-validation fold, minimizing the residual sum of squares between the predicted and true accuracy of each alignment. All results are reported as an average over all of the folds.

The weights given by `Lead-LR` trained on a single fold are in Table 1, sorted by the absolute value of their influence. Note that some of the features have negative weights, this was not allowed in `Facet`. While not intuitive, since all of the features originally trended with accuracy, it can be explained: for instance if an alignment has a high sequence identity but also a high gap density, these two values (because they have opposite signs) will counteract each other. More of the features in `Lead-LR` have significant weight compared with `Facet`, in the previous studies only 5 features ended up being used while here more than 9 features make contributions to the final score.

3.3 An Expanded Parameter Universe

Two *universes* of parameter vectors are used: one containing just over 2000 parameter vectors used for `Facet` called "original", and an extended set of 16,896 parameter vectors called the "extended" universe. The original set was used to construct all of the Oracle advisor sets as well as the `Facet` estimator, the extended set was used for training `Lead-MLP`, `Lead-LR`, and for the Greedy advisor sets.

The extended set was originally developed for use in the `Facet` methodology. It is constructed by first selecting 8 commonly use high-accuracy replacement matrices (`VTML20`, `40`, `80`, `120`, `200`, and `BLOSUM45`, `62`, `80`). For each of the four `Opal` gap penalty parameters (gap-open and gap-extension for both terminal and non-terminal gaps) several possible values were enumerated, and the cross product of these discrete values and the replacement matrices produced a collection (universe) of parameter vectors. Using the default parameter vector as a starting [which was found via inverse parametric alignment; 17, 15] we sampled up to five additional values for each parameter above and below the default. In the case of the terminal gaps, a value relative to the corresponding non-terminal parameter value was chosen rather than a specific penalty. This set constituted approximately 2,000 choices of gap penalty combinations, leading to the number of parameter vectors above in our extended universe.

The original universe is then an informed subsampling of this universe, used to allow the older methods to be solved.

Building on the datasets used in DeBlasio and Kececioglu [8], a carefully curated set of 861 benchmark sequence sets is used consisting of examples from PALI [3] and BENCH [11]; which is itself a collection of alignments from OxBench [26], SABRE [30], and BAliBase [2]. Each benchmark consists of a set of protein sequences as well as a reference alignment which is generally induced using an alignment of the known underlying three-dimensional protein structures. The alignment information is removed to also retrieve the set of constituent sequences. Because the reference is known, the accuracy can be determined as defined earlier.

As mentioned in Section 2 the set of benchmarks is biased toward easy-to-align sets of sequences. This is likely because these benchmark alignments are constructed mostly by hand, and thus only the alignments with which they have the most confidence end up being released. But, in practice, it is expected that sequence sets from across all levels of difficulty will be input. To correct this, each benchmark's ease of alignment was classified using its accuracy using the `Opal` aligner's default parameter vector. This parameter vector was chosen specifically to work best *on average*, and it is the set of parameter values most users will use. The range of accuracies was then divided into 10 equally sized bins and assigned a benchmark to the bin that corresponds to its accuracy when aligned using the default parameter vector. To exemplify the bias in the sample, note that there is a more than 36x increase in the size of the easiest-to-align bin (those with accuracy above 90%, 434) and the hardest-to-align (those $\leq$ 10%, 12). Unless otherwise noted, the weights w_b will be assigned to evenly distribute among *bins* rather than benchmarks. This means with this scheme the accuracy of using only the default parameter vector will be close to 50%, rather than near 80% when weighting by benchmark.

Due to the binning described above, this number of cross validation partitions is somewhat forced since the smallest bins have 12 benchmarks a piece. In each case, each bin was divided randomly into 12 partitions as described earlier. All of the results below show the average across all 12 cross-validation folds.

4 Experimental Results

4.1 Accuracy of the Estimator

Figure 4 shows the comparison of the estimated value (vertical) with the true accuracy (horizontal) for each of the 14 million alignments in the dataset across the two new estimators and `Facet`. Each alignment is shown for when the benchmark it is aligning is in the testing set. To work well as an advisor estimator the value should trend with true accuracy and the plot should have high slope and low spread in order to correctly distinguish between candidates. `Lead-MLP` and `Lead-LR` have both a steeper slope and less spread than `Facet`. The trend-line is shown that is the best linear fit to all of the alignments plotted. The conclusion that `Lead-MLP` and `Lead-LR` outperform `Facet` on the task of matching accuracy is also corroborated by the R-squared values shown, `Lead-MLP` has a value of 0.728 and `Lead-LR` 0.689 while `Facet` is only 0.279. The `Facet`

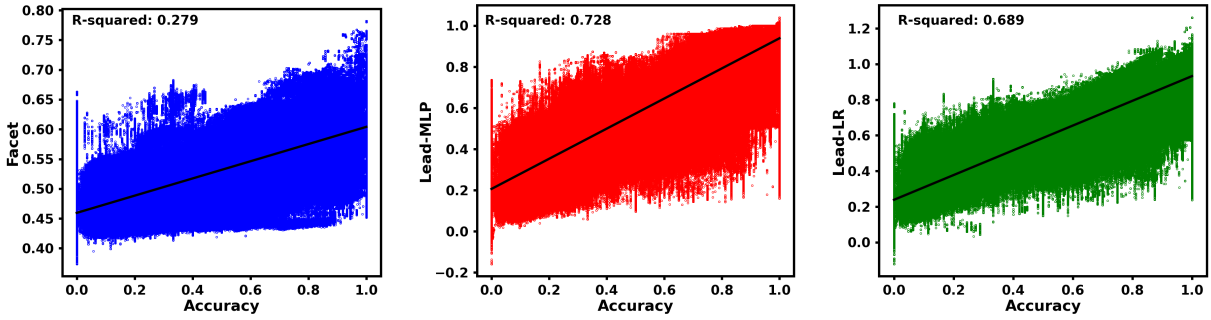

Figure 4: Estimated versus true accuracy of all alignments in the dataset for `Facet`, `Lead-MLP`, and `Lead-LR`.

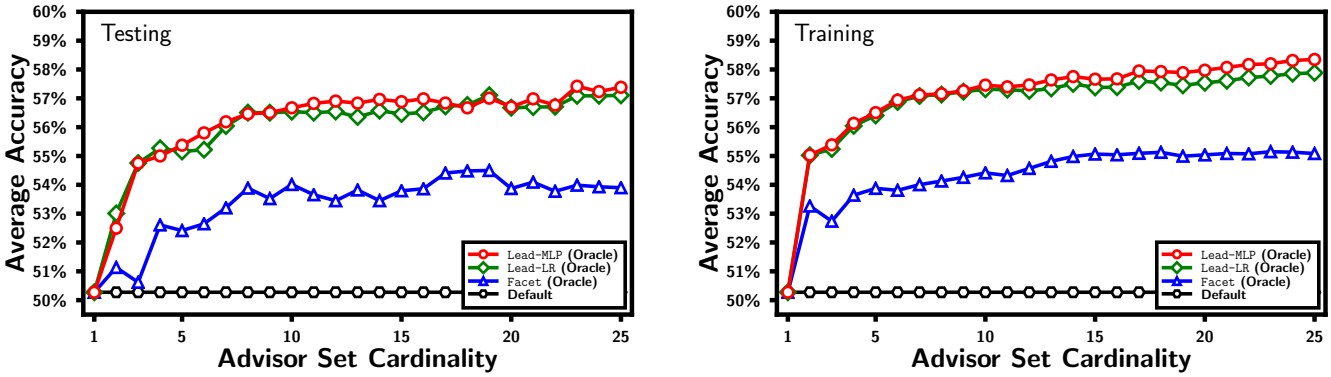

Figure 5: Comparison of the advising accuracy of `Facet`, `Lead-MLP`, and `Lead-LR` using Oracle advisor sets

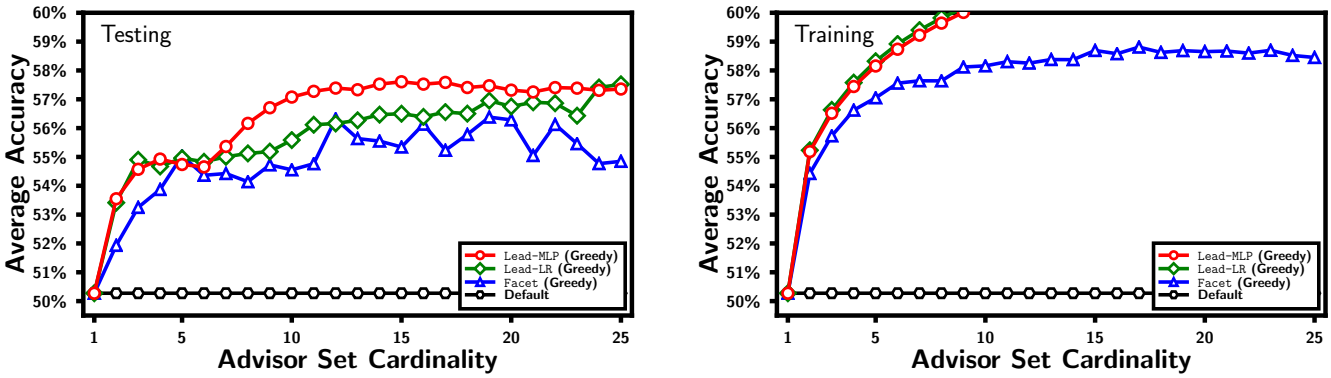

Figure 6: Comparison of the advising accuracy of `Facet` and `Lead-MLP` using Greedy advisor sets

summary R-Squared is very much impacted by the variability in the learned estimators, with a mean of 0.631 across the 12 cross-validation folds compared to 0.737 of `Lead-MLP` and 0.691 of `Lead-LR`.

One contributing factor to the difference seen in the correlation of the estimator and accuracy is that due to the computational limitations at the time, the `Facet` estimator is learned over only parameter pairs in the Oracle set of a given size (in the case of the results above cardinality 25), while because `Lead-MLP` and `Lead-LR` are able to take advantage of modern GPUs, the whole training set was able to be exploited. This change in the training set used in `Facet` also means there is variation in the learned weights at different cardinalities as well as between folds (which will be shown in later sections). As mentioned in Section 3.1 unlike `Facet`, `Lead-MLP` and `Lead-LR` were trained to match the accuracy of a given alignment exactly. Therefore, the objective of `Lead-MLP` and `Lead-LR` estimators is toward increasing R-Squared.

One other item to note in Figure 4 is the scale on the 3 plots. Because there is nothing tying the estimator values of `Facet` to the true accuracies the range of values ends up being quite small. In contrast, by training to the actual accuracy we see estimated accuracies across the whole range from 0 to 1; but this comes at the expense of having predicted values that are somewhat contrary to the definition of accuracy, as you cannot recover fewer than 0 (or more than all) pairs of residues.

4.2 Accuracy of the Advisor

The main purpose of developing `Lead-MLP` and `Lead-LR` was to improve accuracy of the resulting parameter advisor. In the experiments below `Lead-MLP`, `Lead-LR`, and `Facet` are used as the accuracy estimator to construct several advisors. In each case, the increase in accuracy gained by using our new estimators at various advisor set sizes is shown. Advisor-oblivious Oracle sets (using the original universe) and advisor-aware Greedy sets (using the extended universe) are used in combination with the two advisors to create 4 classes of advisors. The `Facet` method relies on a parameter vector set to learn its weights, thus, in all experiments, the `Facet` estimator used was trained on the Oracle set of the specified cardinality, whereas only a single `Lead-MLP` and `Lead-LR` advisor is used for each cross-validation fold.

4.2.1 Oracle Sets

Figure 5 shows the accuracy of the three advisors using Oracle sets on the original universe of parameter vectors as well as accuracy of the default parameter vector. The two plots show the use of the advisor averaged across the 12 cross-validation folds' training (right) and testing (left) benchmarks. Each point on the four lines represents one advisor with its position corresponding to the advisor set cardinality (horizontal) and the advising accuracy (vertical).

`Lead-MLP` and `Lead-LR` both show a higher advising accuracy across all cardinalities both on the training and testing benchmarks. When using Oracle advisor sets on the testing benchmarks `Lead-MLP` shows an average increase of 1.8% accuracy, and `Lead-LR` is similar. Because `Lead-MLP` and `Lead-LR` are trained on a much larger number of parameters as mentioned above, the results show less variation as you increase advisor set cardinality; while not perfectly monotonic it shows less variation than `Facet`. Also, while both accuracy estimation tools eventually plateau (showing little improvement in accuracy with increased cardinality) `Lead-MLP` and `Lead-LR` seem to do this at a larger set size.

With respect to the training benchmarks, `Facet` is trained on only the pairs of alignments from the Oracle set being shown. Previous results had shown that fitting `Facet` to differences in accuracy showed higher advising accuracy, but when a more accurate estimator like `Lead-MLP` or `Lead-LR` is used there is an improvement even over `Facet` which was specifically designed for this task.

4.2.2 Greedy Sets

Figure 6 shows the accuracy of advisors using Greedy advisor sets trained specifically for the individual estimators. Once again, the two plots show the use of the advisor averaged across the 12 cross-validation folds' training (right) and testing (left) benchmarks. Just as above, each point on the four lines represents one advisor with its position corresponding to the advisor set cardinality (horizontal) and the advising accuracy (vertical). While we tested various values of δ [as was done in 8], we found $\delta = 0$ performed best and this was used to produce all figures.

`Lead-MLP` shows consistently higher advising accuracy than the two linear estimators on all but the smallest advisor sets. `Lead-MLP` nor `Lead-LR` benefit as much from Greedy sets as `Facet` with re-

spect to the training benchmarks, seeing an average increase of 3.0% and 3.4% compared to using Oracle sets respectively. While at small cardinalities `Lead-MLP`, `Lead-LR`, and `Facet` show similar performance, across the cardinalities shown the average increase in advising accuracy is still 2.2%. This is mainly due to the fact that the `Facet` estimator is inconsistent in its accuracy as you vary the set cardinality; while `Lead-MLP` and `Lead-LR` continue to improve as the carnality rises.

4.3 Comparison of Alternate Machine Learning Models

Choosing the right algorithm to train a model is crucial. To evaluate the performance of our neural network, we trained models using both decision trees and random forest regressors optimized to minimize squared error. To further improve their performance, we also conducted a local search to find the optimal value for the maximum depth hyperparameter for both algorithms. The maximum depth values tested were 5, 10, 25, 35, and 50. Both the random forest regressor and decision tree regressor converged to the max depth value of 35. The estimated accuracies of these models are shown in Supplemental Figure 1. Both `Lead-MLP` and `Lead-LR` have a higher correlation (R-squared) with true accuracy than any of the other models on the testing set. However, both random forest and decision trees showed an almost perfect correlation with the training data.

The estimators produced with the other machine learning methods were also tested as advisor estimators. The results are shown in Supplemental Figure 2. When this experiment is performed we see just how much the decision tree and random forest regressors overfit to the training benchmarks. On both the Oracle and Greedy advising sets, they have lower performance on the testing set than `Facet` (and in turn `Lead-MLP` and `Lead-LR`) while having training performances that are significantly higher than the other methods.

5 Discussion

In this study, we presented `Lead-MLP` and `Lead-LR`, two new scoring-function-based accuracy estimators that utilize the power of modern machine learning techniques for optimization, rather than combinatorial optimization, to exploit larger datasets. Our results showed that these new estimators have substantially better performance in parameter advising when compared to the `Facet` estimator.

One major issue with multiple sequence alignment tools is the need for user-defined parameter vectors, which can greatly impact downstream analysis if chosen incorrectly. Parameter Advising addresses this challenge by allowing for the selection of input-specific parameter vectors with high accuracy. However, to rank the available alignments, we need to know their true accuracy, but in practice, a reference from which to calculate this is not available. This is where our accuracy estimators, `Lead-MLP` and `Lead-LR`, come in. When used in Parameter Advising, our estimators show an increase of 6% over using only the default parameter vector, which is a 2% increase over using Facet for the same task. Previous studies have also explored different methods to estimate the accuracy of multiple sequence alignments, but our study differs in that we use advanced machine learning techniques to improve the accuracy of the resulting parameter advisor. This highlights the significance of our study as it provides a more efficient and effective method for accuracy estimation in multiple sequence alignment.

We further implemented other machine learning methods to ensure that the choices of models presented were valid. The results seen using random forest and decision tree regressors can be found in the Supplemental Material.

For this specific regression problem, models with a large number of parameters, such as the random forest and decision tree, were prone to overfitting. These models, due to their complexity and excessive number of parameters, tended to fail to generalize to new, unseen data. On the other hand, models with fewer parameters, such as `Lead-MLP` and `Lead-LR` presented in this study, were able to perform well on both the training and unseen data given the limitations of the underlying benchmark sets available.

To keep this work succinct, we did not include results from other estimators such as TCS [4] or MOS [20] on the larger parameter universe of examples described here. Previous studies have shown that for the task of advising, `Facet` is far superior, therefore we are confident that `Lead-MLP` and `Lead-LR` will also outperform them.

5.1 Conclusion

When estimating the accuracy of protein multiple sequence alignments a major improvement can be obtained by exploiting modern machine learning tools. It has been shown in previous studies that when estimating alignment accuracy, it is essential to extract features that help to create estimates for alignments of any size. For parameter advising, carefully constructed small sets of parameter vectors can be very effective, but using large amounts of data and models that can be trained on it shows even better accuracy using the same feature functions. When using the old training methods, along with advisor-agnostic advisor set, you can get an increase of about 4% over using the default parameter vector alone, an increase of 6% is seen when using `Lead-MLP`. Additionally, by training the estimator over the entire set of parameter vectors `Lead-MLP` and `Lead-LR` shows more stability between cross-validation folds and is more generalizable.

5.2 Ongoing Work

We avoided deeper MLP networks after finding that they were overfitting. This can occur when the model has too many parameters, making it able to memorize the training data but not generalize well to new data. On the other hand, shallow MLP networks have fewer parameters and are therefore less likely to overfit. In this specific case, the shallow MLP network was able to generalize better to unseen data and thus performed better on the regression problem. However we are currently exploring more advanced structures such as residual networks [5], and new multiple sequence alignment feature vectors that can be further exploited by deep learning architectures such as transformers [31].

While it should technically be possible, as formulated, finding Oracle Sets on the extended universe is currently intractable. As described in previous studies, the method uses an integer linear program, which was solvable in the original universe despite being technically NP-Complete. When constructed for the extended universe we were unable to find a solution within a week using either CPLEX or Gurobi (the two commonly used tools) on a machine with 256 threads and 2 terabytes of memory. Finding these sets would allow us to use the same universe at all stages of this study.

Acknowledgements

The authors thank Dr. Olac Fuentes and Jose Perez for fruitful discussions and the UTEP Campus Office of Undergraduate Research Initiatives (COURI). This work was partially funded by US Department of Education project number 226150864A, and the Computing Alliance of Hispanic-Serving Institutions (CAHSI) Research Experience for Undergraduates (REU) under NSF Grants CNS-2137791 and HRD-1834620.

References

[1] Virpi Ahola, Tero Aittokallio, Mauno Vihinen, and Esa Uusipaikka. 2008. Model-based prediction of sequence alignment quality. *Bioinformatics* 24, 19 (Sept. 2008), 2165–2171.

[2] Anne Bahr, Julie D Thompson, J C Thierry, and Olivier Poch. 2001. `BAliBASE` (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic Acids Research* 29, 1 (Jan. 2001), 323–326.

[3] S Balaji, S Sujatha, S Sai Chetan Kumar, and N Srinivasan. 2001. `PALI`: a database of Phylogeny and ALIgnment of homologous protein structures. *Nucleic Acids Research* 29, 1 (Jan. 2001), 61–65.

[4] J M Chang, P D Tommaso, and Cedric Notredame. 2014. `TCS`: a new multiple sequence alignment reliability measure to estimate alignment accuracy and improve phylogenetic tree reconstruction. *Molecular Biology and Evolution* 31, 6 (April 2014), 1625–1637.

[5] Dongwei Chen, Fei Hu, Guokui Nian, and Tiantian Yang. 2020. Deep Residual Learning for Nonlinear Regression. *Entropy* 22, 2 (2020), 1–14. `https://doi.org/10.3390/e22020193`

[6] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. 1978. A model of evolutionary change in proteins. *In Atlas of Protein Sequences and Structure* 5 (1978), 345–352.

[7] Dan DeBlasio and John Kececioglu. 2017. Learning Parameter-Advising Sets for Multiple Sequence Alignment. *IEEE/ACM Transactions*

on Computational Biology and Bioinformatics 14, 9 (2017), 1024–1041. Online 2015.

[8] Dan DeBlasio and John Kececioglu. 2018. *Parameter advising for multiple sequence alignment*. Springer International Publishing.

[9] Dan DeBlasio, Kwanho Kim, and Carl Kingsford. 2020. More Accurate Transcript Assembly via Parameter Advising. *Journal of Computational Biology* 27, 8 (2020), 1181–1189. `https://doi.org/10.1089/cmb.2019.0286`

[10] Robert C. Edgar. 2004. `MUSCLE`: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* 32, 5 (2004), 1792–1797.

[11] R C Edgar. 2009. `BENCH`. `http://www.drive5.com/bench`.

[12] S Henikoff and J G Henikoff. 1992. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences USA* 89, 22 (Nov. 1992), 10915–10919.

[13] K Katoh, Kei-ichi Kuma, Hiroyuki Toh, and Takashi Miyata. 2005. `Mafft` version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Research* 33, 2 (Jan. 2005), 511–518.

[14] John Kececioglu and Dan DeBlasio. 2013. Accuracy estimation and parameter advising for protein multiple sequence alignment. *Journal of Computational Biology* 20, 4 (April 2013), 259–279.

[15] John Kececioglu and Eagu Kim. 2006. Simple and Fast Inverse Alignment. In *Proceedings of the 10th Conference on Research in Computational Molecular Biology (RECOMB)*. Springer-Verlag LNBI 7262, 441–455.

[16] John Kececioglu and Dean Starrett. 2004. Aligning alignments exactly. In *Proceedings of the 8th Conference on Research in Computational Molecular Biology (RECOMB)*. ACM, 85–96.

[17] Eagu Kim and John Kececioglu. 2008. Learning Scoring Schemes for Sequence Alignment from Partial Examples. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 5, 4 (2008), 546–556.

[18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

[19] Giddy Landan and Dan Graur. 2007. Heads or tails: a simple reliability check for multiple sequence alignments. *Molecular Biology and Evolution* 24, 6 (2007), 1380–1383.

[20] Timo Lassmann and Erik L L Sonnhammer. 2005. Automatic assessment of alignment quality. *Nucleic Acids Research* 33, 22 (Dec. 2005), 7120–7128.

[21] Jianzhu Ma, Michael Ku Yu, Samson Fong, Keiichiro Ono, Eric Sage, Barry Demchak, Roded Sharan, and Trey Ideker. 2018. Using deep learning to model the hierarchical structure and function of a cell. *Nature Methods* 15 (April 2018), 290–298.

[22] Tobias Müller, Rainer Spang, and Martin Vingron. 2002. Estimating amino acid substitution models: a comparison of Dayhoff's estimator, the resolvent approach and a maximum likelihood method. *Molecular Biology and Evolution* 19, 1 (Jan. 2002), 8–13.

[23] Cedric Notredame, Desmond G Higgins, and J Heringa. 2000. `T-Coffee`: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology* 302, 1 (Sept. 2000), 205–217.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[25] O Penn, Eyal Privman, H Ashkenazy, Giddy Landan, Dan Graur, and Tal Pupko. 2010. `GUIDANCE`: a web server for assessing alignment confidence scores. *Nucleic Acids Research* 38, Web Server (June 2010), W23–W28.

[26] GPS Raghava, Stephen MJ Searle, Patrick C. Audley, Jonathan D. Barber, and Geoffrey J. Barton. 2003. `OXBench`: A benchmark for evaluation of protein multiple sequence alignment accuracy. *BMC Bioinformatics* 4, 1 (2003), 1–23.

[27] F Sievers et al. 2011. Fast, scalable generation of high-quality protein multiple sequence alignments using `Clustal Omega`. *Molecular Systems Biology* 7, 1 (Jan. 2011), 539–539.

[28] J D Thompson, Desmond G Higgins, and Toby J Gibson. 1994. `Clustal W`: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research* 22, 22 (1994), 4673–4680.

[29] J D Thompson, Frédéric Plewniak, Raymond Ripp, Jean-Claude Thierry, and Olivier Poch. 2001. Towards a reliable objective function for multiple sequence alignments. *Journal of Molecular Biology* 314, 4 (Dec. 2001), 937–951.

[30] Ivo Van Walle, Ignace Lasters, and Lode Wyns. 2005. `SABmark`: a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics* 21, 7 (March 2005), 1267–1268.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* (2017), 5998–6008.

[32] L Wang and T Jiang. 1994. On the complexity of multiple sequence alignment. *Journal of Computational Biology : a Journal of Computational Molecular Cell Biology* 1, 4 (1994), 337–348.

[33] Travis J Wheeler and John D Kececioglu. 2007. Multiple alignment by aligning alignments. *Proceedings of the 15th ISCB Conference on Intelligent Systems for Molecular Biology (ISMB), Bioinformatics* 23, 13 (July 2007), i559–i568.

[34] Willard I. Zangwill. 1969. *Nonlinear Programming: A Unified Approach*. Prentice-Hall International, Englewood Cliffs, N.J.